머신러닝 & 딥러닝 4

AI 학술동아리 <MLP>

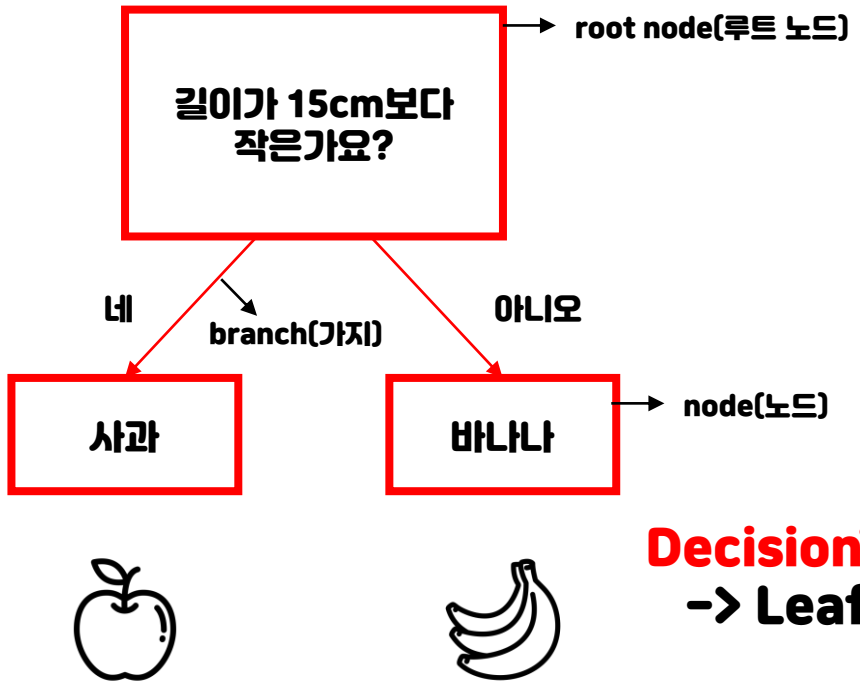
- Index

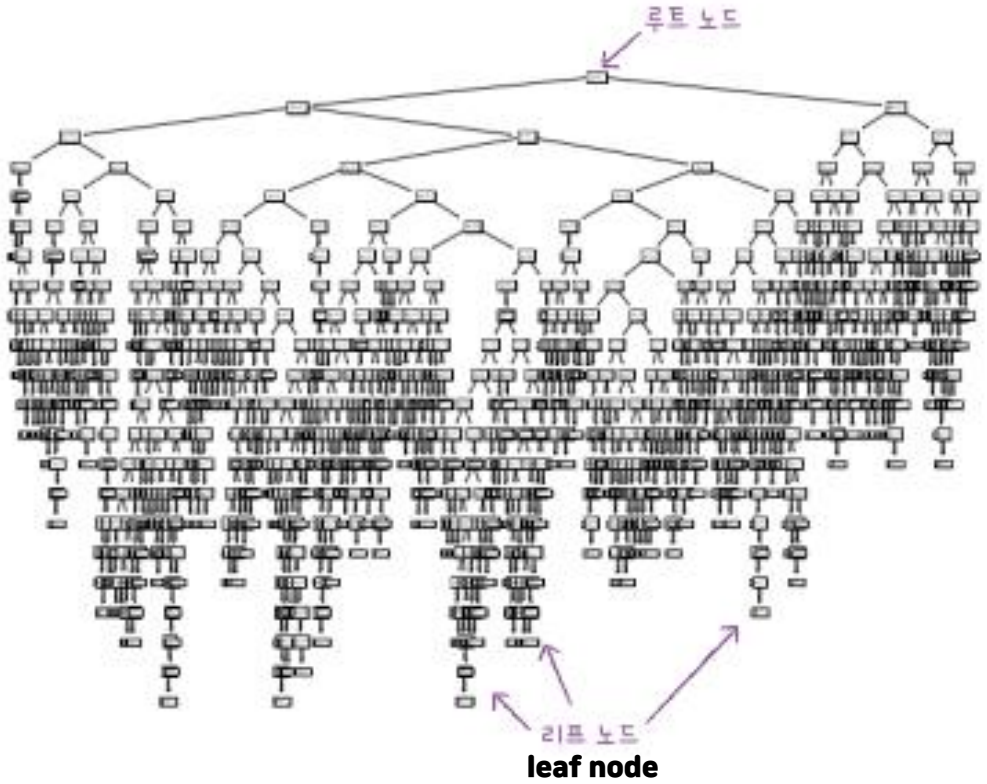
1. 결정 트리
2. 검증 세트 - 교차 검증
3. 하이퍼파라미터 튜닝 - 그리드 서치, 랜덤 서치
4. 앙상블 학습 - 랜덤 포레스트, 엑스트라 트리,
   그레이디언트 부스팅,
   히스토그램 기반 그레이디언트 부스팅

1. Decision Tree(결정 트리)

Decision Tree model은 **스무고개**와 같음

Normalization 과정이 필요 없음





DecisionTreeClassifier

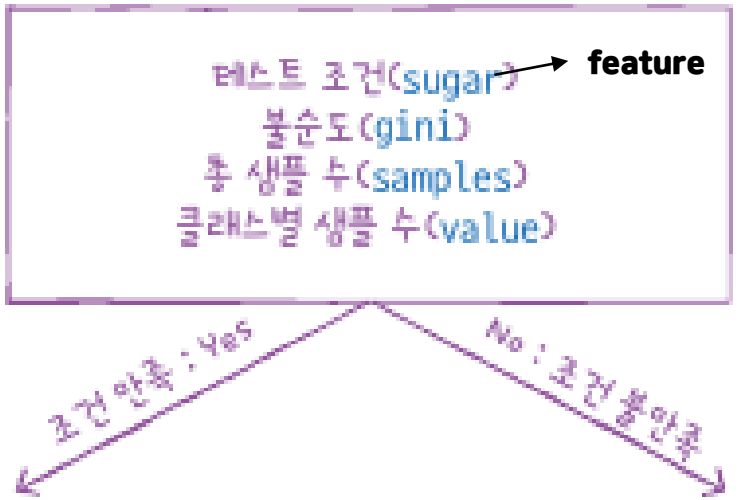
-> Leaf node 에서 가장 많은 클래스가 예측 클래스

DecisionTreeRegressor

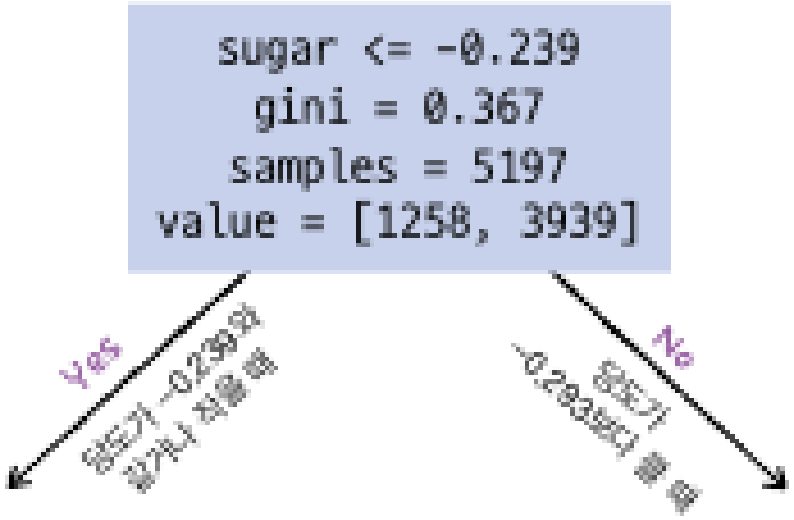
-> Leaf node에 도달한 샘플의 target을 평균한 값을 예측 값으로 사용

1. Decision Tree(결정 트리)

각 Node의 구성



예)



넘어가도 됩니다.

Tip! Impurity(불순도)

DecisionTreeClassifier 클래스의 criterion 매개변수 기본값 = 'gini'

criterion 매개변수 : 데이터를 분할할 기준을 정하는 것

$gini\ impurity = 1 - (\text{음성 } class \text{ 비율}^2 + \text{양성 } class \text{ 비율}^2)$

/ multi class의 경우 각 class의 비율을 제곱해서 더하여 1에서 빼주면 됨

예) 100개 샘플의 비율이 정확히 ½ 씩 일 때

$1 - \left((50/100)^2 + (50/100)^2\right) = 0.5$

예) 100개 샘플이 하나의 class만 있을 때 = 순수 node

$1 - \left((0/100)^2 + (100/100)^2\right) = 0$

information gain(정보 이득) : 부모와 자식 node 사이의 impurity 차이

-> information gain이 최대가 되도록 데이터를 나눔

$information\ gain = \text{부모의 불순도} - (\text{왼쪽 node의 } sample \text{ 수}/\text{부모의 } sample \text{ 수}) \times \text{왼쪽 } node\text{의 } impurity - (\text{오른쪽 node의 } sample \text{ 수}/\text{부모의 } sample \text{ 수}) \times \text{오른쪽 } node\text{의 } impurity$

criterion이 'entropy'일 경우

$entropy\ impurity = -\text{음성 } class \text{ 비율} \times \log_2(\text{음성 } class \text{ 비율}) - \text{양성 } class \text{ 비율} \times \log_2(\text{양성 } class \text{ 비율})$

Tip! Feature importances(특성 중요도)

feature importances의 합은 1

예)
feature : [길이, 무게]
feature importances : [0.7, 0.3]

feature importances의 값이 높은 feature가 가장 유용한 feature

feature importances 계산법
각 node의 information gain과 전체 sample에 대한 비율을 곱한 후 feature별로 더하여 계산

Decision Tree의 feature importances를 feature 선택에 활용 가능

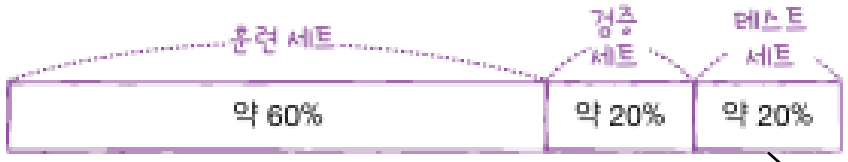
2. Validation Set(검증 세트)

test set를 사용해서 model의 성능을 자꾸 확인하다 보면 점점 test set에 model을 맞추는 것이 됨

test set로 일반화 성능을 올바르게 예측하려면 가능한 한 test set를 사용하지 말아야 함
=> validation set(검증 세트) 사용! (=dev set(개발 세트))

train set에서 model 훈련 -> validation set로 model 평가 => 가장 좋은 model 찾음

가장 좋은 model의 매개변수를 사용해서 train set + validation set 합친 뒤 훈련
-> test set에서 최종 점수 평가



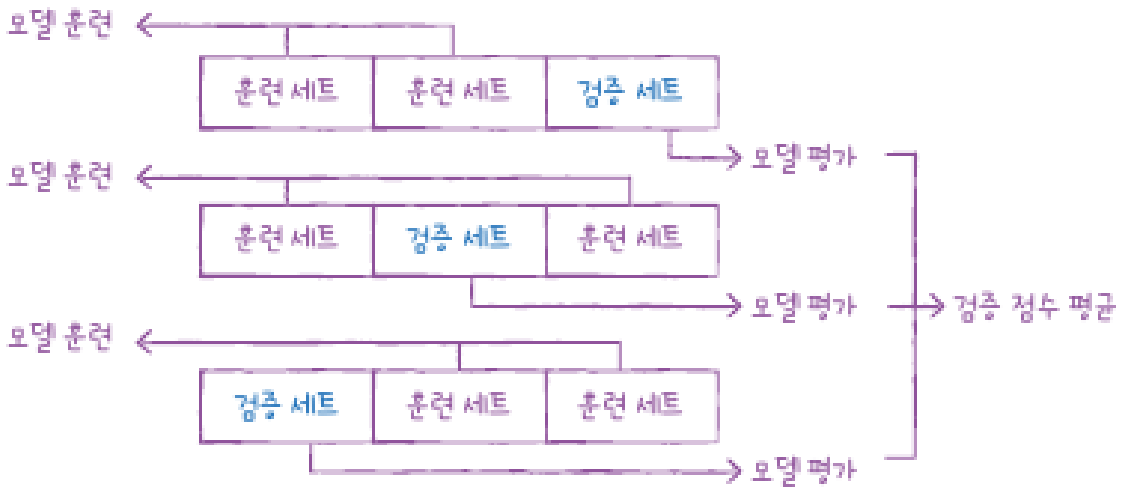
test set는 마지막에 한 번만 사용

2-1. Cross Validation(교차 검증)

validation set을 따로 만들기 때문에 train set가 줄어듦

보통 많은 data를 훈련에 사용할수록 좋은 model이 만들어짐
-> validation set 양을 줄이면 검증 점수가 불안정할 것
=> cross validation(교차 검증) 이용 - 안정적인 검증점수, 훈련에 더 많은 data 사용 가능

3-fold cross validation



k-fold cross validation(k-폴드(겹) 교차 검증)
- train set를 k 부분으로 나눠서 cross validation 수행

train set과 test set를 나눌 때(train_test_split)
전체 데이터를 섞은 후 나눴기 때문에 다시 섞을 필요 없음
- 하지만 굳이 cross validation을 할 때 train set를 섞으려면 splitter(분할기)를 지정해야 함
- Regressor model : splitter = KFold
- Classification model : splitter = StratifiedKFold

3. Hyperparameter Tuning(하이퍼파라미터 튜닝)

Hyperparameter : model이 학습할 수 없어서 사용자가 지정해야만 하는 파라미터

AutoML : 사람의 개입 없이 hyperparameter tuning을 자동으로 수행하는 기술
- 예) **Grid Search**(그리드 서치), **Random Search**(랜덤 서치)

3-1. Grid Search(그리드 서치)

Grid Search = hyperparameter 탐색 + cross validation

Grid Search 과정

1. 먼저 탐색할 매개변수(hyperparameter)를 지정

2. 그 다음 train set에서 grid search를 수행하여 최상의 평균 검증 점수가 나오는
   매개변수 조합을 찾음.
   이 조합은 grid search 객체에 저장됨.

3. grid search는 최상의 매개변수에서 (cross validation에 사용된 train set가 아니라)
   전체 train set를 사용해 최종 model을 훈련.
   이 모델도 grid search 객체에 저장됨.

3-2. Random Search(랜덤 서치)

Random Search = hyperparameter 랜덤하게 탐색 + cross validation

매개변수 값의 목록을 전달하는 것이 아니라 매개변수를 샘플링할 수 있는 확률 분포 객체를 전달
즉, 정해진 값을 전달 X, **값의 범위를 전달해서 랜덤 추출**

이 이외에는 Grid Search와 동일

Tip! Structured Data(정형 데이터) / Unstructured Data(비정형 데이터)

- Structured Data
  - 어떤 구조로 되어있는 Data
  - 프로그래머가 다루는 대부분의 Data
  - CSV, Database, Excel에 저장하기 쉬움
  - 지금까지 배운 ML Algorithm은 structured data에 잘 맞음

- Unstructured Data
  - Database나 Excel로 표현하기 어려운 Data
  - Text Data, Picture, Digital Music
    - 이는 보편적인 사례, NoSQL Database는 CSV나 Excel에 담기 어려운 Text나 JSON Data를 저장하는데 용이
  - Neural Network Algorithm(Deep Learning Algorithm)을 사용

4. Ensemble Learning(앙상블 학습)

Emsemble Learning : 더 좋은 예측 결과를 만들기 위해 여러 개의 model을 훈련하는 ML Algorithm
structured data를 다루는 데 가장 뛰어난 성과를 내는 algorithm
대부분 Decision Tree 기반으로 만들어져 있음

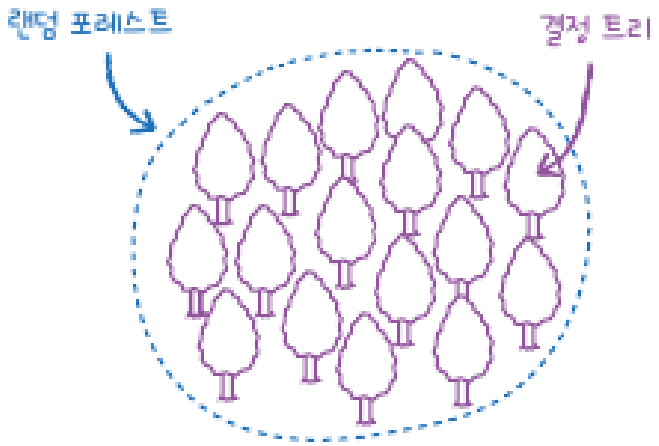
scikit-learn
- Random Forest(랜덤 포레스트) : bootstrap(부트스트랩) sample 사용,
대표 ensemble learning algorithm
- Extra Trees(엑스트라 트리) : Decision Tree의 노드를 랜덤하게 분할
- Gradient Boosting(그레이디언트 부스팅) : 이전 tree의 손실을 보완하는 식으로
얕은 Decision Tree를 연속하여 추가
- Histogram-based Gradient Boosting
(히스토그램 기반 그레이디언트 부스팅) : training data를 256개 정수 구간으로
나누어 빠르고 높은 성능을 냄

그 외 라이브러리
- XGBoost
- LightGBM

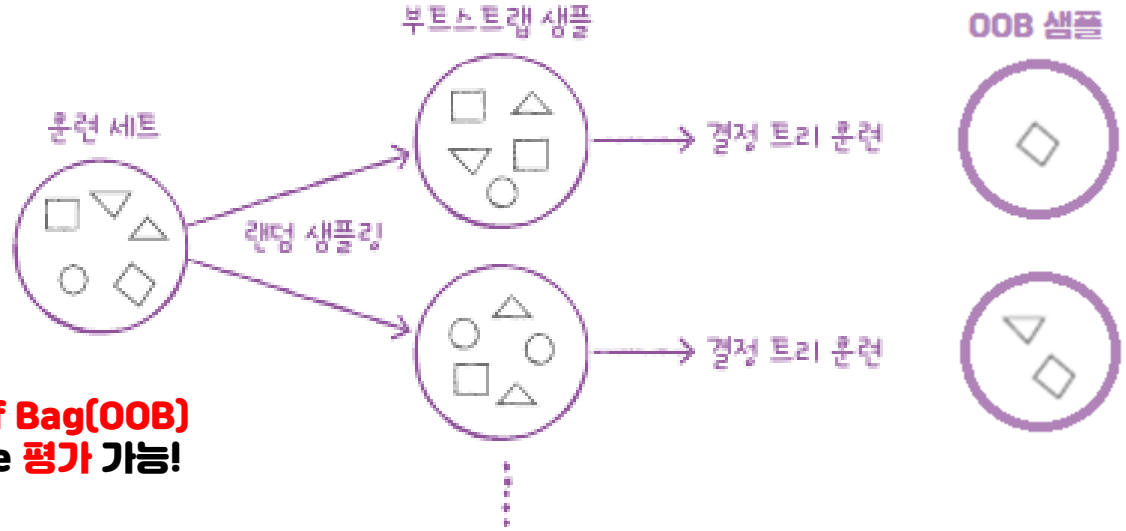
4-1. Random Forest(랜덤 포레스트)

ensemble learning의 대표 주자 중 하나

안정적인 성능 덕분에 널리 사용됨

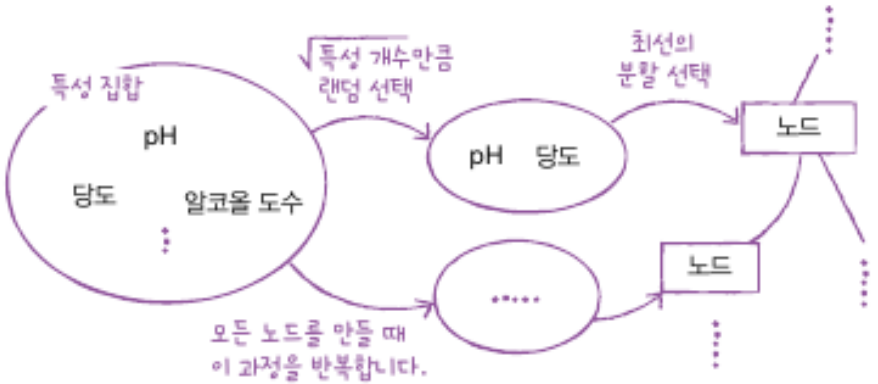


각 Tree를 훈련하기 위한 Data를 랜덤하게 만듦
-> Bootstrap Sample(부트스트랩 샘플) :
Train set에서 중복을 허용하여 data를 랜덤하게 sampling
기본적으로 bootstrap sample은 train set 크기와 같게 만듦



bootstrap sample에 포함되지 않고 남은 sample = Out of Bag(OOB)
OOB 샘플로 각 bootstrap sample로 훈련한 decision tree 평가 가능!

4-1. Random Forest(랜덤 포레스트)



Classification
각 노드를 분할할 때 **feature 개수의 제곱근**만큼 feature 선택
-> 이 중에서 **최선의 분할을 찾음**

Regression
전체 feature 사용

scikit-learn의 random forest는 **기본적으로 100개의 Decision Tree**를 이런 방식으로 **훈련**
-> **Classification** - 각 Tree의 class별 확률을 평균하여 가장 높은 확률을 가진 class로 예측
-> **Regression** – 단순히 각 Tree의 예측을 평균

랜덤하게 선택한 sample과 feature를 사용
-> train set에 **overfitting 방지** 및 validation set과 test set에서 **안정적인 성능**

종종 기본 매개변수 설정만으로 좋은 결과

4-2. Extra Trees(엑스트라 트리)

Random Forest와 비슷

기본적으로 100개의 Decision Tree 훈련
전체 feature 중 일부 feature를 랜덤하게 선택하여 노드 분할

bootstrap sample 사용X
-> **전체 train set 사용**, 노드 분할 시 **무작위 분할**(가장 좋은 분할X)

하나의 Decision Tree에서 feature를 무작위로 분할하면 성능이 낮아짐
-> **많은 tree를 ensemble**하기 때문에 **overfitting을 막고, validation set의 점수를 높임**

보통 Extra Trees가 무작위성이 커서 Random Forest보다 더 많은 Decision Tree 훈련
하지만, 랜덤하게 노드 분할하여 계산 속도가 빠름

4-3. Gradient Boosting(그레이디언트 부스팅)

깊이가 얕은 Decision Tree를 사용하여 이전 tree의 오차를 보완하는 방식으로 ensemble

기본적으로 깊이가 3인 Decision Tree를 100개 사용

깊이가 얕은 Decision Tree를 사용해서 overfitting에 강하고, 일반적으로 높은 일반화 성능을 가짐

gradient descent 방법을 사용하여 tree를 ensemble에 추가
-> Decision Tree를 계속 추가하면서 loss를 줄임
classification - logistic loss function
regression - MSE

일반적으로 gradient boosting이 random forest보다 성능이 높음
하지만, 순서대로 tree를 추가하기 때문에(병렬로 훈련할 수 없기 때문에), 훈련 속도가 느림

learning rate(학습률) 매개변수가 크면 복잡하고 train set에 overfitting된 model을 얻을 수 있음

4-4. Histogram-based Gradient Boosting (히스토그램 기반 그레이디언트 부스팅)

Gradient Boosting의 속도를 개선

Structured Data를 다루는 ML Algorithm 중 가장 인기가 높은 Algorithm

입력 feature를 256개의 구간으로 나눔 -> 노드 분할할 때 최적의 분할을 매우 빠르게 찾을 수 있음

256개의 구간 중 하나를 떼어 놓고 누락된 값을 위해 사용
-> 입력에 누락된 feature가 있더라도 전처리 필요 X

일반적으로 기본 매개변수에서 안정적인 성능

overfitting을 잘 억제하면서 gradient boosting보다 조금 더 높은 성능

Tip! permutation_importance()

feature를 하나씩 랜덤하게 섞어서 model의 성능이 변화하는지 관찰 -> 어떤 특성이 중요한지 계산

train set, test set 모두 적용 가능
estimator(추정기) model에 모두 사용 가능

Tip! XGBoost

scikit-learn이 아닌 Gradient Boosting Algorithm 구현 라이브러리

scikit-learn의 cross_validate() 함수와 함께 사용 가능

다양한 boosting algorithm 지원
-> tree method 매개변수를 'hist'로 지정 시 Histogram-based Gradient Boosting